

# Unison as a Self-Stabilizing Wave Stream Algorithm in Asynchronous Anonymous Networks

Christian Boulinier  
LaRIA, CNRS FRE 2733  
Université de Picardie Jules Verne, France

## Abstract

How to pass from local to global scales in anonymous networks? In such networks, how to organize a self-stabilizing propagation of information with feedback? From Angluin's results, the deterministic leader election is impossible in general anonymous networks. Thus, it is impossible to build a rooted spanning tree. In this paper we show how to use Unison to design a self-stabilizing *barrier synchronization* in an anonymous network. We show that the communication structure of this barrier synchronization designs a self-stabilizing wave stream, or pipelined wave, in anonymous networks. We introduce two variants of waves: Strong Wave and Wavelet. Strong waves can be used to solve the idempotent  $r$ -operator parametrized problem, which implies well known problems like depth-first search tree construction – this instance requires identities for the processors. Wavelets deal with  $\rho$ -distance computation. We show how to use Unison to design a self-stabilizing strong wave stream, and wavelet stream respectively.

**Keywords:** Anonymous Network, Barrier Synchronization, Self-Stabilization, Unison, Wave.

## Correspondance:

Christian BOULINIER  
Email: Christian.Boulinier@u-picardie.fr  
LaRIA, Université de Picardie Jules Verne, Amiens, France  
Tel. +33-322-809-577

# 1 Introduction

Several general message passing problems are useful to achieve many tasks in distributed networks, like broadcasting information, global synchronization, reset, termination detection, or calculation of a global function whose the input depends on several processes or the totality of the processes in the network – see [RH90, Tel94, Lyn96]. In this paper we consider the *wave propagation* problem in asynchronous anonymous networks.

## 1.1 Related Works

In asynchronous systems, there is no global signal. Synchronization is a crucial task. Informally, a synchronizer allows asynchronous systems to simulate synchronous ones. In asynchronous systems, one can at most ensure that no process starts to execute its phase  $i + 1$  before all processes have completed their phase  $i$ . This strongest synchronization task, named *Barrier Synchronization*, was introduced by Misra in [Mis91] in a complete graph. The research about synchronization started with Awerbuch [Awe85]. Communications waves are often used to achieve synchronization. Designing efficient fault-tolerant wave algorithms is an important task. Self-stabilization [Dij74, Dol00] is a general technique to design a system that tolerates arbitrary transient faults, i.e. faults that may corrupt the state of processes or links. [KA98] proposes a self-stabilizing solution for complete graphs. [HL01] designs a solution in uniform rings with an odd size. A relaxed synchronization requirement is defined as follows: the clocks are in phase if the values of two neighboring processes differ by no more than 1, and the clock value of each process is incremented by 1 infinitely often. The *self-stabilizing asynchronous unison* [CFG92] deals with this criterium.

A distributed protocol is *uniform* if every process with the same degree executes the same program. In particular, we do not assume a unique process identifier – the network is anonymous – or some consistent orientation of links in the network such that any dynamic election of a master clock can be feasible. Numerous self-stabilizing wave algorithms use a rooted spanning tree or simply an only initiator, called the *root* – see for instance [Kru79] [ABDT98]. In these cases, protocols are not uniform, they are only at most semi-uniform. So, for a uniform distributed protocol any processor may initiate a wave, and most generally a global computation. Any processor may be an initiator. To face this inherent concurrency, a solution is that every processor maintains the identity of the initiators – see for instance [CDPV02]. That is impossible in an anonymous network.

[KA98] designs a self-stabilizing Barrier Synchronization algorithm in asynchronous anonymous complete networks. For the other topologies the authors use the network with a root, the program is not uniform, but only semi-uniform. An interesting question is to give a solution to this problem in a general connected asynchronous anonymous network. As far as we know, the *phase* algorithm [Tel91] is the only decentralised uniform wave algorithm for a general anonymous network. This algorithm requires that the processors know the diameter, or most simply a common upper bound  $D'$  of the diameter. This algorithm is not self-stabilizing.

## 1.2 Contribution and paper outline

The main task of this paper is to show how Unison can be viewed as a *self-stabilizing wave stream* algorithm in asynchronous anonymous networks scheduled by an *unfair daemon*. The contribution is threefold:

Firstly, we introduce the  $\rho$ -distance barrier synchronization notion. It is a small extension of the barrier synchronization [Mis91] which ensures that no process starts to execute its phase  $i + 1$  before all processes at distance less than or equal to  $\rho$  have completed their phase  $i$ . We show how to design a self-stabilizing *barrier synchronization* at distance  $\rho$  in an anonymous network. The self-stabilizing time complexity is in  $O(n)$  rounds. It has its space complexity in  $O(\log(n) + \log(K))$ , where  $n$  is the number of processes in the network and  $K$  the size of the clock. Secondly, we introduce two variants of Wave: Wavelet and Strong Waves. We show that a strong wave can be used to solve the idempotent  $r$ -operator parametrized problem, and a wavelet deals with  $\rho$ -distance computation. Thirdly, we show that the communication structure of our  $\rho$ -distance barrier synchronization designs a self-stabilizing wavelet stream, or pipelined wavelet, in any anonymous networks. We show that if  $\rho \geq D$  the communications design a self-stabilizing wave stream, and if  $\rho$  is greater than or equal to the length of the longest simple path in the network, then the protocol designs a self-stabilizing strong-wave stream.

The remainder of the paper is organized as follows. In the next section (Section 2), we describe the underlying model for distributed system. We also state what it means for a protocol to be self-stabilizing, we introduce the notion of causal-*DAG* and we present the unison problem and its solutions. In Section 3 we define the  $\rho$ -distance barrier synchronization notion and we introduce a protocol which designs a self-stabilizing *barrier synchronization* at distance  $\rho$  in any anonymous networks. In Section 4 we define two kinds of waves: *wavelet* and *strong waves*, and we show the relationship between a strong wave and the idempotent  $r$ -operator parametrized computation problem. In Section 5, we show how Unison can be view as a wave stream, or a wavelet stream, or a strong wave stream. In Section 6, we give some concluding remarks. Because of the lack of place, some proofs are put back in an annexe.

## 2 Preliminaries

In this section, firstly we define the model of distributed systems considered in this paper, and state what it means for a protocol to be self-stabilizing. Secondly, we present the notions of finite incrementing system and reset on it. Next, we define what a self-stabilizing distributed Unison is.

### 2.1 The model

**Distributed System.** A *distributed system* is an undirected connected graph,  $G = (V, E)$ , where  $V$  is a set of nodes— $|V| = n$ ,  $n \geq 2$ —and  $E$  is the set of edges. Nodes represent *processes*, and edges represent *bidirectional communication links*. A communication link  $(p, q)$  exists iff  $p$  and  $q$  are neighbors. The set of neighbors of every process  $p$  is denoted as  $\mathcal{N}_p$ . The *degree* of  $p$  is the number of neighbors of  $p$ , i.e., equal to  $|\mathcal{N}_p|$ . The distance between two processes  $p$  and  $q$ , denoted by  $d(p, q)$ , is the length of the shortest path between  $p$  and  $q$ . Let  $k$  be a positive integer. Define  $V(p, k)$  as the set of processes such that  $d(p, q) \leq k$ .  $D$  is the diameter of the network.

The program of a process consists of a set of registers (also referred to as variables) and a finite set of guarded actions of the following form:  $\langle \text{label} \rangle :: \langle \text{guard} \rangle \longrightarrow \langle \text{statement} \rangle$ . Each process can only write to its own registers, and read its own registers and registers owned by the neighboring processes. The guard of an action in the program of  $p$  is a boolean expression involving

the registers of  $p$  and its neighbors. The statement of an action of  $p$  updates one or more registers of  $p$ . An action can be executed only if its guard evaluates to true. The actions are atomically executed, meaning the evaluation of a guard and the execution of the corresponding statement of an action, if executed, are done in one atomic step. The *state* of a process is defined by the values of its registers. The *configuration* of a system is the product of the states of all processes. Let a distributed protocol  $\mathcal{P}$  be a collection of binary transition relations denoted by  $\mapsto$ , on  $\mathcal{C}$ , the set of all possible configurations of the system.  $\mathcal{P}$  describes an oriented graph  $S = (\mathcal{C}, \mapsto)$ , called the *transition graph* of  $\mathcal{P}$ . A sequence  $e = \gamma_0, \gamma_1, \dots, \gamma_i, \gamma_{i+1}, \dots$  is called an *execution* of  $\mathcal{P}$  iff  $\forall i \geq 0, \gamma_i \mapsto \gamma_{i+1} \in S$ . A process  $p$  is said to be *enabled* in a configuration  $\gamma_i$  ( $\gamma_i \in \mathcal{C}$ ) if there exists an action  $A$  such that the guard of  $A$  is true in  $\gamma_i$ . The value of a register  $r$  of a process  $p$  in the state  $\gamma_i$ , is denoted by  $p^i.r$ .  $i$  is the moment of the state  $\gamma_i$ . When there is no ambiguity, we will omit  $i$ . Similarly, an action  $A$  is said to be enabled (in  $\gamma$ ) at  $p$  if the guard of  $A$  is true at  $p$  (in  $\gamma$ ). We assume that each transition from a configuration to another is driven by a *distributed scheduler* called *daemon*. In this paper, we consider only an Asynchronous distributed Daemon. The *Asynchronous Daemon* chooses any nonempty set of enabled processes to execute an action in each computation step (Unfair Daemon).

In order to compute the time complexity, we use the definition of *round* [DIM97]. This definition captures the execution rate of the slowest processor in any computation. Given an execution  $e$ , the *first round* of  $e$  (let us call it  $e'$ ) is the minimal prefix of  $e$  containing the execution of one action of the protocol or the neutralization of every enabled processor from the first configuration. Let  $e''$  be the suffix of  $e$ , i.e.,  $e = e'e''$ . Then *second round* of  $e$  is the first round of  $e''$ , and so on.

**Self-Stabilization.** Let  $\mathcal{X}$  be a set. A *predicate*  $P$  is a function that has a Boolean value—**true** or **false**—for each element  $x \in \mathcal{X}$ . A predicate  $P$  is *closed* for a transition graph  $S$  iff every state of an execution  $e$  that starts in a state satisfying  $P$  also satisfies  $P$ . A predicate  $Q$  is an attractor of the predicate  $P$ , denoted by  $P \triangleright Q$ , iff  $Q$  is closed for  $S$  and for every execution  $e$  of  $S$ , beginning by a state satisfying  $P$ , there exists a configuration of  $e$  for which  $Q$  is true. A transition graph  $S$  is *self-stabilizing* for a predicate  $P$  iff  $P$  is an attractor of the predicate **true**, i.e., **true**  $\triangleright P$ .

## 2.2 Causal DAGs

**Definition 2.1 (Events and Causal DAGs)** Let  $\gamma_{t_0}\gamma_{t_0+1}\dots$  be a finite or infinite execution.  $\forall p \in V, (p, t_0)$  is an event. Let  $\gamma_t \rightarrow \gamma_{t+1}$  be a transition. If the process  $p$  executes a guarded action during this transition, we say that  $p$  executes an action at time  $t+1$ , and we say that  $(p, t+1)$  is an event or a  $p$ -event. The causal DAG associated is the smallest relation  $\leadsto$  on the set of events that satisfies:

1. Let  $(p, t)$  be an event with  $t > t_0$ . Let  $t'$  be the greatest integer such that  $t_0 \leq t' < t$  and  $(p, t')$  is an event, then  $(p, t') \leadsto (p, t)$
2. Let  $(p, t)$  be an event and let  $t > t_0$ . Let  $q \in \mathcal{N}_p$  and let  $t'$  be the greatest integer such that  $t_0 \leq t' < t$  and such that  $(q, t')$  is an event, then  $(q, t') \leadsto (p, t)$ .

The causal order  $\preceq$  on the set of events is the reflexive and transitive closure of the causal relation  $\leadsto$ . The past cone of an event  $(p, t)$  is the causal-DAG induced by every event  $(q, t')$  such that  $(q, t') \preceq (p, t)$ . A past cone involves a process  $q$  iff there is a  $q$ -event in the cone. The cover of an event  $(p, t)$  is the set of processes  $q$  covered by the past cone of  $(p, t)$ , this set is denoted by  $Cover(p, t)$ .

**Definition 2.2 (Cut)** A cut  $C$  on a causal DAG is a map from  $V$  to  $\mathbb{N}$ , which associates each process  $p$  with a time  $t_p^C$  such that  $(p, t_p^C)$  is an event. We mix this map with its graph:  $C = \{(p, t_p^C), p \in V\}$ . The past of  $C$  is the events  $(p, t)$  such that  $t \leq t_p^C$ . It is denoted by  $]\leftarrow, C]$ . The future of  $C$  is the events  $(p, t)$  such that  $t_p^C \leq t$ . It is denoted by  $[C, \rightarrow]$ . A cut is coherent if  $(q, t') \preceq (p, t)$  and  $(p, t) \preceq (p, t_p^C)$  then  $(q, t') \preceq (q, t_q^C)$ . A cut  $C_1$  is less than or equal to a cut  $C_2$ , denoted by  $C_1 \preceq C_2$ , if the past of  $C_1$  is included in the past of  $C_2$ . If  $C_1$  and  $C_2$  are coherent and  $C_1 \preceq C_2$  then  $[C_1, C_2]$  is the induced causal DAG defined by the events  $(p, t)$  such that  $(p, t_{p_1}^{C_1}) \preceq (p, t) \preceq (p, t_{p_2}^{C_2})$ . Any segment  $[C_1, C_2]$  is a sequence of events, each event of  $C_1$  is called an initial event.

### 2.3 Distributed Unison

*Unison*, or most precisely *Self-Stabilizing Asynchronous Unison*, is a relaxed *self-stabilizing Barrier Synchronization* in the following meaning: the clocks are in phase if the values of two neighboring processes differ by no more than 1, and the clock value of each process is incremented by 1 infinitely often. Self-stabilizing Unison was introduced by [CFG92]. There is a possibility of deadlock if the size of the clock is too short— see [BPV04]. A little algebraic framework, and some vocabulary are necessary. The vocabulary will be used in the definition of the algorithm 1.

**Algebraic framework** Let  $\mathbb{Z}$  be the set of integers and  $K$  be a strictly positive integer. Two integers  $a$  and  $b$  are said to be *congruent modulo  $K$* , denoted by  $a \equiv b[K]$  if and only if  $\exists \lambda \in \mathbb{Z}, b = a + \lambda K$ . We denote  $\bar{a}$  the unique element in  $[0, K - 1]$  such that  $a \equiv \bar{a}[K]$ .  $\min(\bar{a} - \bar{b}, \bar{b} - \bar{a})$  is a *distance* on the torus  $[0, K - 1]$  denoted by  $d_K(a, b)$ . Two integers  $a$  and  $b$  are said to be *locally comparable* if and only if  $d_K(a, b) \leq 1$ . We then define the *local order relationship*  $\leq_l$  as follows:  $a \leq_l b \stackrel{\text{def}}{\iff} 0 \leq \bar{b} - \bar{a} \leq 1$ . If  $a$  and  $b$  are two locally comparable integers, we define  $b \ominus a$  as follows:  $b \ominus a \stackrel{\text{def}}{=} \bar{b} - \bar{a}$  if  $a \leq_l b$  then  $\bar{b} - \bar{a}$  else  $-\bar{a} - \bar{b}$ . If  $a_0, a_1, a_2, \dots, a_{p-1}, a_p$  is a sequence of integers such that  $\forall i \in \{0, \dots, p-1\}, a_i$  is locally comparable to  $a_{i+1}$ , then  $S = \sum_{i=0}^{p-1} (a_{i+1} \ominus a_i)$  is the *local variation* of this sequence.

**Incrementing system** We define  $\mathcal{X} = \{-\alpha, \dots, 0, \dots, K - 1\}$ , where  $\alpha$  is a positive integer. Let  $\varphi$  be the function from  $\mathcal{X}$  to  $\mathcal{X}$  defined by:  $\varphi(x) \stackrel{\text{def}}{=} \bar{x} + 1$  if  $x \geq 0$  then  $\bar{x} + 1$  else  $x + 1$ . The pair  $(\mathcal{X}, \varphi)$  is called a *finite incrementing system*.  $K$  is called the *period* of  $(\mathcal{X}, \varphi)$ . Let  $\text{tail}_\varphi = \{-\alpha, \dots, 0\}$  and  $\text{stab}_\varphi = \{0, \dots, K - 1\}$  be the sets of “extra” values and “expected” values, respectively. The set  $\text{tail}_\varphi^*$  is equal to  $\text{tail}_\varphi \setminus \{0\}$ . We assume that each process  $p$  maintains a clock register  $r_p$  with an incrementing system  $(\mathcal{X}, \varphi)$ . Let  $\gamma$  the system configuration, we define the predicate  $WU$ :

$$WU(\gamma) \stackrel{\text{def}}{=} \forall p \in V, \forall q \in \mathcal{N}_p : (r_p \in \text{stab}_\varphi) \wedge (d(r_p, r_q) \leq 1) \text{ in } \gamma.$$

**Intrinsic Path Delay [BPV04]** Let  $\gamma$  a configurations in  $WU$ , the clock values of neighboring processes are locally comparable. We define the four notions:

**Delay** The delay along a path  $\mu = p_0 p_1 \dots p_k$ , denoted by  $\Delta_\mu$ , is the local variation of the sequence  $r_{p_0}, r_{p_1}, \dots, r_{p_k}$ , i.e,  $\Delta_\mu = \sum_{i=0}^{k-1} (r_{p_{i+1}} \ominus_l r_{p_i})$  if  $k > 0$ , 0 otherwise ( $k = 0$ ).

**Intrinsic Delay** The delay between two processes  $p$  and  $q$  is *intrinsic* if it is independent on the choice of the path from  $p$  to  $q$ . The delay is *intrinsic* iff it is *intrinsic* for every  $p$  and  $q$  in  $V$ . In this case, and at time  $t$ , the intrinsic delay between  $p$  and  $q$  is denoted by  $\Delta_{(p,q)}$ .

**WU<sub>0</sub>** The predicate  $WU_0$  is true for a system configuration  $\gamma$  iff  $\gamma$  satisfies  $WU$  and the delay is intrinsic in  $\gamma$ .

**Precedence relationship** When Delay is intrinsic, it defines a total preordering on the processes in  $V$ , named *precedence relationship*. This relationship depends on the state  $\gamma \in WU_0$ . The absolute value of the delay between two processes  $p$  and  $q$ , is equal to or less than the distance  $d(p, q)$  in the network. This remark is important for the following.

**Cyclomatic Characteristic  $C_G$  [BPV04]** If  $G$  is an acyclic graph, then its cyclomatic characteristic  $C_G$  is equal 2. Otherwise  $G$  contains cycles: Let  $\Lambda$  be a cycle basis, the length of the longest cycle in  $\Lambda$  is denoted  $\lambda(\Lambda)$ . The cyclomatic characteristic of  $G$ , is equal to the lowest  $\lambda(\Lambda)$  among cycle bases. It follows from the definition of  $C_G$  that  $C_G \leq 2D$ .

**Unison Definition** We assume that each process  $p$  maintains a register  $p.r \in \chi$ . The self-stabilizing *asynchronous (distributed) unison* problem, or most shortly the *unison* problem, is to design a uniform protocol so that the following properties are true in every execution [BPV05]:

**Safety** :  $WU$  is closed. **Synchronization**: In  $WU$ , a process can increment its clock  $r_p$  only if the value of  $r_p$  is lower than or equal to the clock value of all its neighbors. **No Lockout (Liveness)**: In  $WU$ , every process  $p$  increments its clock  $r_p$  infinitely often. **Self-Stabilization** :  $\Gamma \triangleright WU$ .

The following guarded action solves the *synchronization property* and the *safety*:

$$\forall q \in \mathcal{N}_p : (r_q = r_p) \vee (r_q = \varphi(r_p)) \longrightarrow r_p := \varphi(r_p); \quad (1)$$

The predicate  $WU_0$  is closed for any execution of this guarded action. Moreover, for any execution starting from a configuration in  $WU_0$ , the *no lockout property* is guaranteed. Generally this property is not guaranteed in  $WU$ . A few general schemes to self-stabilizing the non-stabilizing protocols have been proposed. The first self-stabilizing asynchronous unison was introduced in [CFG92]. The deterministic protocol proposed needs  $K \geq n^2$ . The stabilization time complexity is in  $O(nD)$ . The second solution is proposed in [BPV04]. The authors show that if  $K$  is greater than  $C_G$  then  $WU = WU_0$  and the *no lockout property* is guaranteed in  $WU$ . (see Definition 2.3). The protocol is self-stabilizing if  $\alpha \geq T_G - 2$ , where  $T_G$  is the length of the longest chordless cycle (2 in tree networks). One can notice that  $C_G$  and  $T_G$  are bounded by  $n$ . So, even if  $C_G$  and  $T_G$  are unknown, we can choose  $K \geq n + 1$  and  $\alpha = n$ . Its self-stabilizing time complexity is in  $O(n)$ . In [BPV06], the authors present the Protocol  $WU\_Min$ , which is self-stabilizing to asynchronous unison in at most  $D$  rounds in trees.

### 3 Barrier Synchronization

#### 3.1 Barrier synchronization at distance $\rho$

*Barrier Synchronization problem* has been specified in [KA98]. Let  $\rho$  be an integer greater than 0. The relaxation of this problem at distance  $\rho$  is the following. Let  $K$  be an integer greater than 1. We assume that each process  $p$  maintains a  $K$ -order clock register  $p.R \in \{0, 1, \dots, K - 1\}$ . Each process executes a cyclique sequence of  $K$  terminating phases (the critical section  $\langle\langle cs \rangle\rangle$ ). The following two properties are required for each phase:

**Global Unison (Safety) :** for each phase  $x \in \{0, \dots, K-1\}$ , no process  $p$  can proceed to phase  $\overline{x+1}$  until all nodes  $q$ , such that  $d(p, q) \leq \rho$ , has executed its phase  $x$ .

**No lockout (liveness):** every process increments its clock infinitely often.

For  $\rho = 1$ , this specification is the specification of the standard stabilized unison. For  $\rho \geq D$ , this specification is the specification of the global Barrier Synchronization.

### 3.2 The general self-stabilizing Scheme

The idea is to stabilize an underlayer unison in order to synchronize a  $\delta K$ -clock, with  $\delta$  large enough to guarantee that the absolute value of the delay between every two processes at distance less than or equal to  $\rho$  is never larger than  $\delta$ . It is sufficient that  $\delta \geq \rho$  holds. We take  $\chi = \{-\alpha, \dots, 0, \dots, \delta K - 1\}$  and  $\alpha \geq T_G - 2$ . We use the unison of [BPV04] which stabilizes in  $O(n)$ . The protocol is describe in Algorithm 1. To ensure self-stabilization in  $WU_0$ , we require  $\delta K > C_G$ . If we want to program a Barrier Synchronization, we must take  $\delta \geq D$ , thus from  $C_G \leq 2D$ , if  $K \geq 3$  then the inequality  $K\delta > C_G$  holds. In the remainder we suppose that the inequality  $K\delta > C_G$  holds.

---

**Algorithm 1** (*SS – WS*) Self-Stabilizing  $\rho$ -Barrier Synchronization algorithm for the process  $p$

---

**Constant and variable:**

$\mathcal{N}_p$ : the set of neighbors of process  $p$ ;  $p.r \in \chi$ ;

**Boolean Functions:**

$ConvergenceStep_p \equiv p.r \in tail_\varphi^* \wedge (\forall q \in \mathcal{N}_p : (q.r \in tail_\varphi) \wedge (p.r \leq_{tail_\varphi} q.r))$ ;  
 $LocallyCorrect_p \equiv p.r \in stab_\varphi \wedge (\forall q \in \mathcal{N}_p, q.r \in stab_\varphi \wedge ((p.r = q.r) \vee (p.r = \varphi(q.r)) \vee (\varphi(p.r) = q.r)))$ ;  
 $NormalStep_p \equiv p.r \in stab_\varphi \wedge (\forall q \in \mathcal{N}_p : (p.r = q.r) \vee (q.r = \varphi(p.r)))$ ;  
 $ResetInit_p \equiv \neg LocallyCorrect_p \wedge (p.r \notin init_\varphi)$ ;

**Actions:**

$NA : NormalStep_p \longrightarrow \text{if } p.r \equiv \rho - 1[\rho] \text{ then } \langle\langle CS\ 2 \rangle\rangle \text{ else } \langle\langle CS\ 1 \rangle\rangle ; p.r := \varphi(p.r)$ ;  
 $CA : ConvergenceStep_p \longrightarrow p.r := \varphi(p.r)$ ;  
 $RA : ResetInit_p \longrightarrow p.r := \alpha \text{ (reset)}$ ;

---

### 3.3 Analysis

**Lifting construction** In order to analyse the protocol 1 we introduce for each process  $p$ , a global device, the register  $\widetilde{p.r}$ . Of course the value of this virtual register is inaccessible to the process  $p$ . Informally  $\widetilde{p.r}$  is a way to unwind of the register  $p.r$ . Let  $\gamma_{t_0} \gamma_{t_0+1} \dots$  be an infinite execution starting in  $WU_0$ . Let  $p_0$  be a maximal process, according to the *precedence relation* – see Remark 2.3 – for the state  $\gamma_0$ . Let  $\perp_0 = p_0.r$  at time 0. For each process  $p \in V$ , we unwind the register  $p.r$  in the following manner. We associate a virtual register  $\widetilde{p.r}$ . For the state  $\gamma_0$ , we initiate this virtual register by the instruction  $\widetilde{p.r} := \perp_0 + \Delta_{(p_0, p)}^0$ . During the execution, for each transition  $\gamma_t \rightarrow \gamma_{t+1}$  the instruction  $\widetilde{p.r} := \widetilde{p.r} + 1$  holds if and only if  $p.r := \overline{p.r + 1}$  holds during the same transition. For  $k \geq \perp_0$  we define the cut  $C_k = \{(p, t_{p,k}), p \in V\}$  where  $t_{p,k}$  is the smallest time such that  $\widetilde{p.r} := k$ . The first question is to prove that this cuts are coherent. We first introduce the easy lemma:

**Lemma 3.1** *If  $(p, t) \rightsquigarrow (q, t')$  then:  $\widetilde{q^{t'}.r} \in \{\widetilde{p^t.r}, \widetilde{p^t.r} + 1\}$ . Inductively, if  $(q_0, t_0) \rightsquigarrow (q_1, t_1) \rightsquigarrow (q_2, t_2) \dots \rightsquigarrow (q_i, t_i)$  then:  $\widetilde{q_i^{t_i}.r} \in \{\widetilde{q_0^{t_0}.r}, \dots, \widetilde{q_0^{t_0}.r} + i\}$*

From the Lemma 3.1, if  $(q, t) \preceq (p, t_{p,k})$  then  $(q, t) \preceq (q, t_{q,k})$ . It follows the proposition:

**Proposition 3.2** *For every  $k \geq \perp_0$  the cut  $C_k$  is coherent.*

**Virtual register  $p.R$  and virtual clock** For each process  $p$  we associate the register  $p.R$ , which is virtual. Its value is evaluated by the procedure: if  $p.r \in \text{stab}_\rho$  then  $p.R := p.r/\delta$  else  $p.R := -1$ , where the symbol  $/$  is the *integer division* operator. The virtual register  $p.R$  defines a clock on  $\{-1, 0, \dots, K-1\}$ . The algorithm 1 solves self-stabilizing Asynchronous Unison, so every process  $p$  increments its clock  $p.r$  infinitely often. We deduce that  $p.R$  increments infinitely often, thus:

**Lemma 3.3 (Liveness)** *For every process  $p$ , the virtual register  $p.R$  is incremented infinitely often. Consequently  $\langle\langle \text{CS } 2 \rangle\rangle$  is executed infinitely often.*

**Theorem 3.4** *If  $\delta \geq \rho$ , once the protocol is stabilized, it solves the Barrier Synchronization at distance  $\rho$  for the virtual clock defined by the register  $p.R$ .*

**Proof.** We consider the phase  $U = [C_{U\delta}, C_{U\delta+\delta-1}]$ , for any event  $(p, t)$  in this sequence, the register  $p.R$  is equal to  $\overline{U}[K]$ . Let  $p$  and  $q$  be two processes, such that  $d(p, q) \leq \rho$ . Let  $(p, t_p)$  and  $(q, t_q)$  be in  $C_{U\delta+\delta}$ . Suppose that  $t_p \leq t_q$ , at time  $t_p$  the register  $\widetilde{q.r} \in \{U\delta + \delta - i, i \in \{0, \dots, \rho - 1\}\}$ , thus at time  $t_p$ , the critical section  $\langle\langle \text{CS } 2 \rangle\rangle$  of the phase  $U$  is terminated for the process  $q$ .  $\square$

Our protocol synchronizes processors at distance  $\rho$  in any anonymous general network. On the general graph, this synchronizer does not need any identity and does not build any real or virtual spanning tree. Here, the broadcast runs in the beginning of a phase from any decentralised node  $p$ . For each node  $q \in V(p, \rho)$ , at the end of the phase for  $q$ , the node knows that information is gone to all the others nodes in  $V(q, \rho)$ , the feedback is implicit. The time complexity of a phase  $[C_{U\delta}, C_{U\delta+\delta-1}]$  is  $\delta$  rounds in worst case. The message complexity is  $2\delta |E|$ , which is the price to pay for uniformity. But is this message complexity usable? We will give a positive answer.

## 4 Wavelet, Wave and Strong Wave

During each phase of Algorithm 1, the structure of communications is a kind of wave depending of the value of  $\delta$ . These communication structures are formally defined in this section. In the section 5, following the Theorem 5.2, we will be able to use these communications to compute some important functions on the network, for instance an infimum if  $\delta \geq D$ , or most generally the idempotent  $r$ -operator parametrized calculation problem when  $\delta \geq n$ , and so to solve many silent tasks [Duc98].

### 4.1 Walk and Wave

**Definition 4.1 Walk.** A Walk is a finite non empty word  $m = q_0 q_1 \dots q_r$  on the alphabet  $V$ , such that for all  $i \in \{0, r-1\}$ ,  $q_i = q_{i+1}$  or  $q_{i+1} \in \mathcal{N}_{q_i}$ . A walk is circular if  $r > 1$  and  $q_0 = q_r$ . The walk  $m$  is beginning in  $q_0$  denoted  $\text{head}(m)$ , and is ending in  $q_r$ . Its length is  $r$ .

Let  $m$  be a walk, if there exists two words  $m_1$  and  $m_2$ , and a circular walk  $u$  such that  $m = m_1 u m_2$ , ( $u$  is a factor of  $m$ ), then  $m_1 \text{head}(u) m_2$  is a walk and we write:  $m \rightarrow m_1 \text{head}(u) m_2$ . The transitive closure of the relationship  $\rightarrow$  defines a strict partial ordering  $\xrightarrow{*}$  in the set of walks. A simple walk is a minimal walk according to the  $\xrightarrow{*}$  partial ordering. Most simply, a simple walk is a walk without any repetition. An elementary walk is a walk such that if for  $i < j$ ,  $q_i = q_j$  then for all  $k \in \{i, \dots, j\}$ ,  $q_k = q_i$ . A reducing of a walk  $m$  is a simple walk  $m'$  such that  $m \xrightarrow{*} m'$ .

**Walk cover of an event in a sequence.** Let  $S = [C_1, C_2]$  be a sequence of events. If in  $S$ ,  $(q, t') \preceq (p, t)$  then there exists a causality chain from  $(q, t')$  to  $(p, t)$ :  $(q, t') = (q_0, t_0) \rightsquigarrow (q_1, t_1) \rightsquigarrow$



$(q_2, t_2) \dots \rightsquigarrow (q_r, t_r) = (p, t)$  , its associated walk is the walk  $q_0 q_1 \dots q_r$ . The walk cover of an event  $(p, t) \in S$  is the set of walks associated to the causality chains of  $S$  ending to  $(p, t)$ . This set is denoted by  $\text{WalkCover}(p, t)$ . Of course, this set contains the walk of length 0 denoted by  $p$ .

**Lemma 4.2** *If  $m \in \text{WalkCover}(p, t)$  then there exists an elementary walk  $m'$  in  $\text{WalkCover}(p, t)$  such that  $m \xrightarrow{*} m'$*

**Proof.** Let  $m = q_0 q_1 \dots q_r$  the associated walk of the causality chain  $(q_0, t_0) \rightsquigarrow (q_1, t_1) \rightsquigarrow (q_2, t_2) \dots \rightsquigarrow (q_r, t_r)$ . Suppose that  $m = m_1 u m_2$  where  $u$  is a circular walk  $q_i q_i \dots q_j$ . From the definition of  $\rightsquigarrow$  relationship, there exists a chain:  $(q_i, t_i) \rightsquigarrow (q_i, t_{i_1}) \rightsquigarrow (q_i, t_{i_2}) \dots \rightsquigarrow (q_i, t_j)$ . Let  $l$  the length of this chain. If  $v = \prod_{k=1}^l q_i$  then  $\bar{m} = m_1 v m_2$  is an element of  $\text{WalkCover}(p, t)$ . Such a rewriting operation is possible only a finite number of times, at the end, the word is elementary.  $\square$

**Definition 4.3 (Wavelet, Wave, and Strong Wave)** *Following [Tel94], we assume that there are special events called decide events, the nature of these events depends of the algorithm. Let  $k$  an integer. A  $k$ -wavelet is a sequence of events  $[C_1, C_2]$  that satisfies the following two requirements:*

*The causal DAG induced by  $[C_1, C_2]$  contains at least one decide event.*

*For each decide event  $(p, t)$  , the past of  $(p, t)$  in  $[C_1, C_2]$  covers  $V(p, k)$ .*

*We simply call it a wave when  $k \geq D$ , where  $D$  is the diameter of the network.*

*A strong wave is a wave  $[C_1, C_2]$  that satisfies the following added requirement:*

*For each decide event  $(p, t)$  in  $[C_1, C_2]$ , and for each simple walk  $m_0 = q_0 q_1 \dots q_{n-1} p$  ending in  $p$ , there exists a causality chain  $(q_0, t_0) \rightsquigarrow (q'_1, t_1) \dots \rightsquigarrow (q'_{r-1}, t_{r-1}) \rightsquigarrow (p, t)$  in  $[C_1, C_2]$ , such that its associated walk  $m$  is elementary, and  $m \xrightarrow{*} m_0$ .*

## 4.2 Infima and $r$ -operators

Tel, in his work about wave algorithms [Tel94], introduces the infimum operators. An infimum  $\oplus$  over a set  $\mathbb{S}$ , is an associative, commutative and idempotent (i.e.  $x \oplus x = x$ ) binary operator. If  $P = \{a_1, a_2, \dots, a_r\}$  is a finite part of  $(S)$  then, from the associativity,  $\oplus P$  makes sense as  $a_1 \oplus a_2 \oplus \dots \oplus a_r$ . And if  $a \in S$ , then  $a \oplus P$  makes sense as  $a \oplus a_1 \oplus a_2 \oplus \dots \oplus a_r$ . Such an operator defines a partial order relation  $\leq_\oplus$  over  $\mathbb{S}$ , by  $x \leq_\oplus y$  if and only if  $x \oplus y = x$ . We suppose that  $\mathbb{S}$  has a greater element  $e_\oplus$ , such that  $x \leq_\oplus e_\oplus$  for every  $x \in \mathbb{S}$ . Hence  $(\mathbb{S}, \oplus)$  is an Abelian idempotent semi-group with  $e_\oplus$  as identity element for  $\oplus$ . Ducourthial introduces in [Duc98] the notion of  $r$ -operator which generalizes the infimum operators.

**Definition 4.4** *The binary operator  $\triangleleft$  on  $\mathbb{S}$  is a  $r$ -operator if there exists a  $(\mathbb{S}, \oplus)$ -endomorphism  $r$ , called  $r$ -function, such that:  $\forall x, y \in \mathbb{S}, x \triangleleft y = x \oplus r(y)$ . Let  $\triangleleft$  be a  $r$ -operator on  $\mathbb{S}$ , and let  $r$  be its associated  $r$ -function ,  $\triangleleft$  is idempotent if and only if:  $\forall x \in \mathbb{S}, x \leq_\oplus r(x)$ . A mapping  $\triangleleft$  from  $(\mathbb{S})^n$  to  $\mathbb{S}$  is an  $n$ -ary  $r$ -operator if there exists  $n - 1$   $(\mathbb{S}, \oplus)$ -endomorphisms  $r_1, r_2, \dots, r_{n-1}$  such that for all  $(x_0, x_1, \dots, x_{n-1}) \in (\mathbb{S})^n$  :  $\triangleleft(x_0, x_1, \dots, x_{n-1}) = x_0 \oplus r_1(x_1) \oplus \dots \oplus r_{n-1}(x_{n-1})$*

**Remark 4.5**  *$r$  is an endomorphism, which means that for all  $x, y$  in  $\mathbb{S}$ ,  $r(x \oplus y) = r(x) \oplus r(y)$ . From the definition of  $\leq_\oplus$ , we deduce that  $r$  is compatible with  $\leq_\oplus$ , formally:  $\forall x, y \in \mathbb{S}, x \leq_\oplus y \Rightarrow r(x) \leq_\oplus r(y)$*

### 4.3 Infimum and $r$ -operator parametrized computation problem

Let  $[C_1, C_2]$  be a wave . We denote by  $\mathcal{N}_p^t$  the set of processes such that there exists a time  $t_q$  such that  $(q, t_q) \rightsquigarrow (p, t)$ . Note that  $p$  may be in  $\mathcal{N}_p^t$ . Because of the lack of place, the proof of Theorem 4.10 is in the annexe .

**Infimum computation** Give each process  $p$ , an extra variable  $p.res : \mathbb{S}$  . Each register  $p.res$  is initialised during the initial event of  $p$  by the value  $p.v_0$ . let  $(p, t)$  be any event in  $[C_1, C_2]$ . Whenever  $(p, t)$  holds,  $p.res$  is set to the value  $p.v_0 \oplus \{q^{t_q}.res, q \in \mathcal{N}_p^t\}$ . Tel shows the following theorem:

**Theorem 4.6** [Tel94] *A wave can be used to compute an infimum.*

**Idempotent  $r$ -operator parametrized computation problem** Let  $[C_1, C_2]$  be a strong wave . We associate to each oriented link  $(p_i, p_j)$  of  $G = (V, E)$  a idempotent  $r$ -function:  $r_{p_i, p_j}$ . By extention, for the sequence  $(p_i, p_i)$  we associate the identity:  $r_{ii} = id$ . Like above, give each process  $p$ , an extra constant  $p.v_0 : \mathbb{S}$  and a register  $p.res$ . Each register  $p.res$  is initialised during the initial event of  $p$  by the value  $p.v_0$ . let  $(p, t)$  be any event in  $[C_1, C_2]$ . Whenever  $(p, t)$  holds,  $p.res$  is set to the value  $p.v_0 \oplus \{r_{q,p}(q.res), q \in \mathcal{N}_p^t\}$ . Each node  $p$  can be seen as a  $(d + 1)$ -ary  $r$ -operator if  $d$  is the degree of the node.

**Definition 4.7** *For any walk  $\mu = p_0 p_1 \dots p_n$ , we define  $eval(\mu) = r_\mu(p_0.v_0)$  , with  $r_\mu = r_{p_{n-1}, p_n} \circ r_{p_{n-2}, p_{n-1}} \circ \dots \circ r_{p_0, p_1}$ , where  $\circ$  is the composition of functions. For any  $p \in V$ , the sets  $\Lambda'_p$  and  $\Lambda_p$  are defined by:  $\Lambda'_p = \{eval(\mu), \mu \in \Sigma'_p\}$  and  $\Lambda_p = \{eval(\mu), \mu \in \Sigma_p\}$ , where  $\Sigma'_p$  is the set of the walks ending to  $p$ , and  $\Sigma_p$  is the set of the simple walks ending to  $p$ .*

From the definitions and the idempotence of the  $r$ -operators, the following lemma holds:

**Lemma 4.8** *Assume that  $m$  and  $m'$  are two walks with  $p = head(m)$ . We suppose that  $m \xrightarrow{*} m'$ . Then  $r_m(p.v_0) \geq_\oplus r_{m'}(p.v_0)$ , and if  $m$  is elementary then  $r_m(p.v_0) = r_{m'}(p.v_0)$*

**Definition 4.9 (Legitimate output)** *We define the legitimate output of a process  $p$  as the quantity:  $\oplus \Lambda_p$*

**Theorem 4.10** *A strong wave can be used to solve the idempotent  $r$ -operator parametrized problem.*

## 5 Unison as a self-stabilizing wave stream algorithm, applications

### 5.1 Analysis of the Unison Behavior starting in $WU_0$

**Lemma 5.1** *Let  $k \geq \perp_0$ . If  $(p, t)$  is an event in the interval  $[C_k, \rightarrow[$ , then:  $V(p, \widetilde{p^t.r} - k) \subset Cover(p, t)$  and  $\Sigma_p^{\widetilde{p^t.r} - k} \subset WalkCover(p, t)$ .*

*Where  $\Sigma_p^\rho$  is the set of simple walks of length less than or equal to  $\rho$ , ending to  $p$ .*

**Proof.** The lemma is true for the initial events of  $[C_k, \rightarrow[$ . Let  $\mathcal{A}$  be the set of events  $(p, t)$  in  $[C_k, \rightarrow[$  such that the sentence:

$$V(p, \widetilde{p^t.r} - k) \subset Cover(p, t) \wedge \Sigma_p^{\widetilde{p^t.r} - k} \subset WalkCover(p, t)$$

does not hold. We assume that  $\mathcal{A}$  is not empty, let  $(q, \tau)$  a minimal event in  $\mathcal{A}$  according to  $\preceq$ . Let  $\delta = \widetilde{q^\tau.r} - k$ , and let  $p_1 \in V(q, \delta)$ . If  $p_1 = q$  then  $p_1 \in \text{Cover}(q, \tau)$ , else there exists  $q_1 \in \mathcal{N}_q$  such that  $p_1 \in V(q_1, \delta - 1)$ .  $(q, \tau)$  is not a initial event, so  $q_1 \in \mathcal{N}_q^\tau$  and there exists  $\tau_{q_1}$  such that  $(q_1, \tau_{q_1}) \rightsquigarrow (q, \tau)$ , and by the minimality of  $(q, \tau)$  the inclusion  $V(q_1, \delta - 1) \subset \text{Cover}(q_1, \tau_{q_1})$  holds and thus  $V(q_1, \delta - 1) \subset \text{Cover}(q, \tau)$  and  $p_1 \in \text{Cover}(q, \tau)$ . Following the same way, let  $m$  be a walk in  $\Sigma_p^\delta$ , if  $m = q$  then  $m \in \text{WalkCover}(q, \tau)$ , else if  $m = p_1 p_2 \dots p_r q$  then  $p_r \in \mathcal{N}_q^\tau$  because  $(q, \tau)$  is not a initial event, so there exists  $\tau_{p_r}$  such that  $(p_r, \tau_{p_r}) \rightsquigarrow (q, \tau)$ , and by the minimality of  $(q, \tau)$  the inclusion  $\Sigma_{p_r}^{\delta-1} \subset \text{WalkCover}(p_r, \tau_{p_r})$  holds, and thus  $p_1 p_2 \dots p_r q \in \text{WalkCover}(q, \tau)$ . So  $(q, \tau)$  is not in  $\mathcal{A}$ . Thus  $\mathcal{A} = \emptyset$ , and the lemma is proved.  $\square$

As corollary, we deduce the important following theorem:

**Theorem 5.2** *Let  $k \geq \perp_0$  and  $\delta$  be a positive integer, then  $[C_k, C_{k+\delta}]$ , with  $C_{k+\delta}$  as the set of decide events, is a  $\delta$ -wavelet, and a wave if  $\delta \geq D$ . If  $\delta$  is greater than or equal to the length of a longest simple walk in  $G$ , then  $[C_k, C_{k+\delta}]$  is a strong wave.*

## 5.2 Self-stabilizing computation of an infimum at distance $\rho$

**If  $\rho \geq D$**  For each process  $p$ , the registers  $p.v_0$  and  $p.res$  are initialized by the same value. We need one step for the initialization, and  $D$  steps for the wave of calculation. So we take  $\delta \geq D + 1$ . For any integer  $U$ ,  $[C_{U\delta}, C_{U\delta+\delta-1}]$  is a wave. So we define the critical sections as follows:

$\ll CS2 \gg \equiv$  initialization of  $p.v_0$  and  $p.res$

$\ll CS1 \gg \equiv p.res := p.v_0 \bigoplus \{q.res, q \in \mathcal{N}_p\}$

From Theorem 4.6, at the cut  $C_{U\delta+\delta-1}$  the register  $p.res$  contains the right value  $\bigoplus \{q.v_0, q \in V\}$ .

**If  $\rho < D$**  We take  $\delta = \rho + 1$ . We suppose that the register  $q.v_0$  is initialised during the critical section  $\ll CS2 \gg$  at the beginning of the phase, precisely when the register  $p.r$  takes the value  $U\delta$ . To reach the objective, we define for each process  $p$  two added registers  $p.v_1$  and  $p.v_2$ . These two registers are initialized at the date  $C_{U\delta}$  during the critical section  $\ll CS2 \gg$ , by the value  $p.v_0$ . For  $\alpha \in \{1, 2, \dots, \rho\}$ , at the date  $C_{U\delta+\alpha}$ , the action  $\ll CS1 \gg$  is the following:

$p.v_1 := p.v_2; p.v_2 := p.v_0 \bigoplus \{q.v_{\varphi(q)}, q \in \mathcal{N}_p\}$

with, if  $q.r = p.r$  then  $\varphi(q) = 2$ , and if  $q.r = p.r + 1$  then  $\varphi(q) = 1$ .

**Theorem 5.3** *At the cut  $C_{U\delta+\delta-1}$  the register  $p.res$  contains the right value:  $\bigoplus \{q.v_0, q \in V(p, \rho)\}$ .*

The proof is in the annexe.

## 6 Concluding remarks

We showed how the structure of the communications between processes of Unison can be viewed as a wave stream. Thanks to this structure, we have been able to build a self-stabilizing wave stream algorithm in asynchronous anonymous networks scheduled by an unfair daemon. Precisely, we showed that the behavior of Unison can be viewed as a self-stabilizing wave,  $k$ -wavelet or bidirected link flood streams. From these remarks, in any asynchronous anonymous network scheduled by an unfair daemon, we deduced self-stabilizing solutions to the barrier synchronization problem, the infimum calculation problem, and the idempotent  $r$ -operator parametrized calculation problem. Now, an important question would be to reduce the self-stabilizing time complexity of unison from  $O(n)$  to  $O(D)$  in a general graph.

## References

- [ABDT98] L. O. Alima, J. Beauquier, A. K. Datta, and S. Tixeuil. Self-stabilization with global rooted synchronizers. In *IEEE 18th International Conference on Distributed Computing Systems (ICDCS 98)*, pages 102–109, 1998.
- [Awe85] B. Awerbuch. Complexity of network synchronization. *Journal of the Association of the Computing Machinery*, 32(4):804–823, 1985.
- [BPV04] C Boulinier, F Petit, and V Villain. When graph theory helps self-stabilization. In *PODC '04: Proceedings of the twenty-third annual ACM symposium on Principles of distributed computing*, pages 150–159, 2004.
- [BPV05] C Boulinier, F Petit, and V Villain. Synchronous vs. asynchronous unison. In *7th Symposium on Self-Stabilizing Systems (SSS'05)*, LNCS 3764, pages 18–32, 2005.
- [BPV06] C Boulinier, F Petit, and V Villain. Toward a time-optimal odd phase clock unison in trees. In Springer-Verlag, editor, *Eighth International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS 06)*, Lecture Notes in Computer Science, 2006.
- [CDPV02] A Cournier, AK Datta, F Petit, and V Villain. Snap-stabilizing PIF algorithm in arbitrary networks. In *IEEE 22nd International Conference on Distributed Computing Systems (ICDCS 02)*, pages 199–206. IEEE Computer Society Press, 2002.
- [CFG92] JM Couvreur, N Francez, and M Gouda. Asynchronous unison. In *Proceedings of the 12th IEEE International Conference on Distributed Computing Systems (ICDCS'92)*, pages 486–493, 1992.
- [Dij74] EW Dijkstra. Self stabilizing systems in spite of distributed control. *Communications of the Association of the Computing Machinery*, 17:643–644, 1974.
- [DIM97] S Dolev, A Israeli, and S Moran. Uniform dynamic self-stabilizing leader election. *IEEE Transactions on Parallel and Distributed Systems*, 8(4):424–440, 1997.
- [Dol00] S Dolev. *Self-Stabilization*. The MIT Press, 2000.
- [Duc98] B. Ducourthial. New operators for computing with associative nets. In *The 5th International Colloquium On Structural Information and Communication Complexity Proceedings (SIROCCO'98)*, pages 51–65. Carleton University Press, 1998.
- [HL01] S.T. Huang and T.J. Liu. Phase synchronization on asynchronous uniform rings with odd size. In *IEEE Transactions on Parallel and Distributed System*, volume 12(6), pages 638–652, 2001.
- [KA98] S. Kulkarni and A. Arora. Low-cost fault-tolerance in barrier synchronizations. In *ICPP: 27th International Conference on Parallel Processing*, 1998.
- [Kru79] HSM Kruijer. Self-stabilization (in spite of distributed control) in tree-structured systems. *Information Processing Letters*, 8:91–95, 1979.
- [Lyn96] N. Lynch. *Distributed algorithms*. Morgan Kaufmann, Chichester, UK, 1996.
- [Mis91] J Misra. Phase synchronization. *Information Processing Letters*, 38(2):101–105, 1991.
- [RH90] M. Raynal and JM Helary. *Synchronization and control of distributed Systems and programs*. John Wiley and Sons, Chichester, UK, 1990.
- [Tel91] G. Tel. *Topics in Distributed Algorithms*. Cambridge University Press, 1991.
- [Tel94] G. Tel. *Introduction to Distributed Algorithms*. Cambridge University Press, 1994.

## 7 Annexe

### 7.1 Proof of Theorem 4.10

**Proposition 7.1** *For any  $p \in V$ ,  $\oplus \Lambda'_p$  exists, and the equality  $\oplus \Lambda'_p = \oplus \Lambda_p$  holds.*

**Proof.**

$\Lambda_p$  is finite, so  $\oplus \Lambda_p$  exists, furthermore  $\Lambda_p \subset \Lambda'_p$ . If  $m = p_0 p_1 \dots p_n$  be a not elementary walk, there exists a simple walk  $m'$  such that  $m \xrightarrow{*} m'$ . From the lemma 4.8,  $r_m(p.v_0) \geq_{\oplus} r_{m'}(p.v_0)$  and  $r_{m'}(p.v_0) \geq_{\oplus} \oplus \Lambda_p$  hold. The proposition follows.

□

**Lemma 7.2** *Let  $(p, t)$  be an event in  $[C_1, C_2]$ , then at time  $t$ ,*

$$p^t.res = \bigoplus \{eval(\mu), \mu \in WalkCover(p, t)\}$$

**Proof.**

Let  $\mathcal{A}$  be the set of events  $(p, t)$  in  $[C_1, C_2]$  such that the equality is not true. Note that the minimal events in  $[C_1, C_2]$  are not in  $\mathcal{A}$ . If  $\mathcal{A}$  is empty, the proof is finished. Suppose that  $\mathcal{A}$  is not empty. Let  $(p, t)$  a minimal event of  $\mathcal{A}$  according to the relation  $\preceq$ :

$$p^t.res := p.v_0 \bigoplus \{r_{q,p}(q^{t_q}.res), q \in \mathcal{N}_p^t\}$$

But, by definition:  $WalkCover(p, t) = \bigcup_{q \in \mathcal{N}_p^t} \{\mu p, \mu \in WalkCover(q, t_q)\} \cup \{p\}$ .

From the minimality of  $(p, t)$  in  $\mathcal{A}$ , the events  $(q, t_q)$  are not in  $\mathcal{A}$ , so:

$$p^t.res = p.v_0 \bigoplus_{q \in \mathcal{N}_p^t} r_{qp} \left( \bigoplus \{eval(\mu), \mu \in WalkCover(q, t_q)\} \right)$$

But  $r_{pq}$  is compatible with  $\leq \oplus$  (remark 4.5), thus:

$$p^t.res = p.v_0 \bigoplus_{q \in \mathcal{N}_p^t} \{r_{qp}(eval(\mu)), \mu \in WalkCover(q, t_q)\}$$

$(p, t)$  is not an initial event, so  $p \in \mathcal{N}_p^t$  and:

$$p.v_0 \geq \oplus \{r_{pp}or_\mu(head(\mu).v_0), \mu \in WalkCover(p, t_p)\}$$

We deduce, from associativity of  $\oplus$  and from  $r_{qp}or_\mu = r_{\mu p}$  that :

$$p^t.res = \bigoplus_{q \in \mathcal{N}_p^t} \{eval(\mu p), \mu \in WalkCover(q, t_q)\}$$

but  $WalkCover(p, t) = \bigcup_{q \in \mathcal{N}_p^t} \{\mu p, \mu \in WalkCover(q, t_q)\}$ , so:

$$p^t.res = \{eval(\mu), \mu \in WalkCover(p, t)\}$$

We deduce that  $(p, t)$  is not in  $\mathcal{A}$ , which is a contradiction. We deduce that  $\mathcal{A} = \emptyset$  and the lemma. □

**Theorem 7.3 (4.10)** *A strong wave can be used to solve the idempotent  $r$ -operator parametrized problem.*

**Proof.** If  $(p, t)$  is a decide event then, from the Lemma 7.2, we establish that  $p^t.res = \{eval(\mu), \mu \in WalkCover(p, t)\}$  holds, and that  $WalkCover(p, t)$  satisfies the Definition 4.3. Recall that  $\Lambda_p = \{eval(\mu), \mu \in \Sigma_p\}$ . For any  $m \in WalkCover(p, t)$ , there exists  $m_0 \in \Sigma_p$  such that  $m \xrightarrow{*} m_0$  and from the Lemma 4.8 the inequality  $eval(m) \geq eval(m_0)$  holds. We deduce that  $p^t.res \geq \oplus \Lambda_p$ . Conversely, if  $m_0 \in \Lambda_p$ , there exists  $m \in WalkCover(p, t)$  such that  $m \xrightarrow{*} m_0$ , but from the Lemma 4.2, there exists also a walk  $m_1 \in WalkCover(p, t)$  such that  $m \xrightarrow{*} m_1$  and  $m_1 \xrightarrow{*} m_0$ , and from Lemma 4.2  $eval(m_1) = eval(m_0)$ . We deduce that  $p^t.res \leq \oplus \Lambda_p$ . From these two inequalities, we deduce  $p^t.res = \oplus \Lambda_p$  and the theorem is proved. □

## 7.2 Proof of Theorem 5.3

**Proposition 7.4** *For  $p \in V$  and  $\alpha \in \{1, \dots, \rho\}$ , at the date  $C_{U\delta+\alpha}$ , hold the equalities:*

$$p.v_1 = \bigoplus \{q.v_0, q \in V(p, \alpha - 1)\} \text{ and } p.v_2 = \bigoplus \{q.v_0, q \in V(p, \alpha)\}$$

**Proof.**

At the date  $C_{U\delta}$ , any process  $p$  satisfies  $p.v_1 = p.v_0$  and  $p.v_2 = p.v_0$ , it is the initializing step. Let  $\mathcal{A}$  the set of events in  $[C_{U\delta+1}, C_{U\delta+\delta-1}]$ , for which the proposition is not true. We assume that  $\mathcal{A}$  is not empty. Let  $(p, t)$  a minimal event in  $\mathcal{A}$ . let  $\alpha \in \{1, 2, \dots, \rho\}$  such that  $(p, t) \in C_{U\delta+\alpha}$ . There exists  $t_0$  such that  $(p, t_0) \rightsquigarrow (p, t)$ . We have  $p^t.v_1 = p^{t_0}.v_2 =$  and  $p^{t_0}.v_2 = \bigoplus \{q.v_0, q \in V(p, \alpha - 1)\}$ . This equality is true even if  $\alpha = 1$ . Now,  $p^t.v_2 = p.v_0 \bigoplus \{q^{t_q}.v_{\varphi(q)}, q \in \mathcal{N}_p\}$ . From the minimality of the event  $(p, t)$ , the events  $(q, t_q)$ , where  $t_q < t$ , are not in  $\mathcal{A}$  and are in  $[C_{U\delta}, C_{U\delta+\delta-1}]$ . So,  $p.v_0 \bigoplus \{q^{t_q}.v_{\varphi(q)}, q \in \mathcal{N}_p\} = \bigoplus \{q.v_0, q \in V(p, \alpha)\}$ . We obtain a contradiction. We deduce that  $\mathcal{A}$  is empty, and the proposition follows □

As corollary, we obtain the theorem:

**Theorem 7.5 (5.3)** *At the cut  $C_{U\delta+\delta-1}$  the register  $p.res$  contains the right value  $\bigoplus \{q.v_0, q \in V(p, \rho)\}$ .*